Module : Programmation JAVA

Enseignant: Semchedine Moussa

Email : mchemsedine@yahoo.fr

Niveau : 3ème LMD math professionnelle

Volume : 2 cours + 1 TD + 1 TP

Mode d'évaluation : 2/3 examen écrit + 1/3 TP (préparation+assiduité+interrogation+ présence)

Année : 2014-2015

Objectifs :

Il s'agit de familiariser l'étudiant sur l'approche de Programmation Orientée Objet, en lui donnant un langage pouvant répondre à tous les principes de la POO

1. Algorithmique

- Un algorithme est une suite finie d'instructions permettant de résoudre un problème
- L'algorithme doit être :
  Lisible : compréhensible même par un non informaticien
  Précis : utiliser le minimum d'instructions
  Structuré : composé de différentes parties facilement identifiables
  Complet : il doit tenir compte de tous les cas possibles

2. La programmation

- Un programme est une séquence d'instructions écrites dans un langage de programmation, et exécutées automatiquement par une machine
- Un langage de programmation est un code de communication permettant à un être humain de dialoguer avec la machine (Php, Pascal, Fortran, C, C++, Java, Caml ….)

Remarque : la différence entre les langages de programmation réside dans plusieurs aspects : la syntaxe, la gestion de la mémoire, les types prédéfinis, le domaine d'utilisation

3. Les niveaux de programmation

- **Langages bas niveau**

Un langage de programmation est dit de bas niveau lorsque le codage de celui-ci se rapproche du langage machine (dit "binaire"), Le langage **machine** et le langage **assembleur** sont considérés comme des langages de bas niveau.

- **Langages évolués (haut niveau)**

Un langage évolué est un langage très proche du langage humain (plus simple à utiliser, mais permet généralement moins d'actions sur les ressources systèmes)

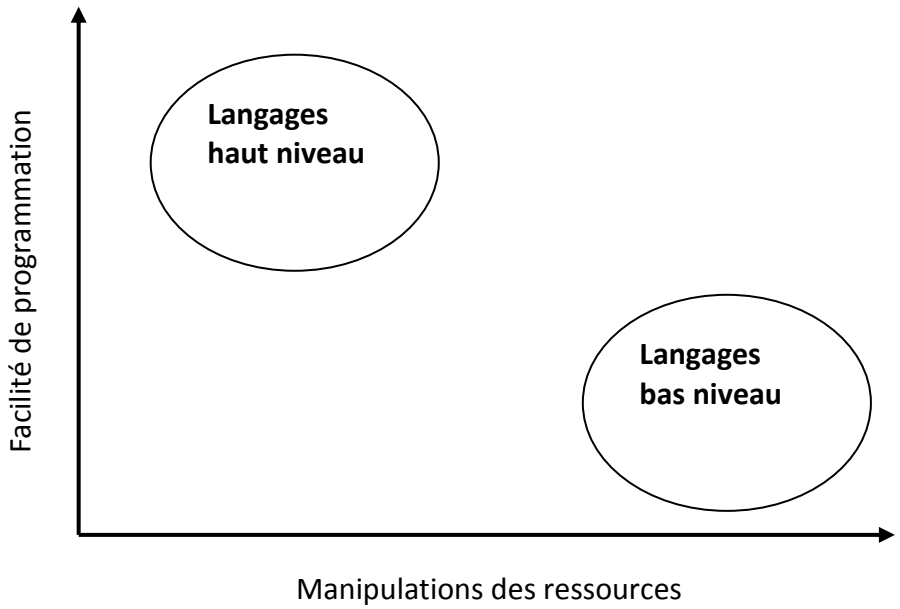
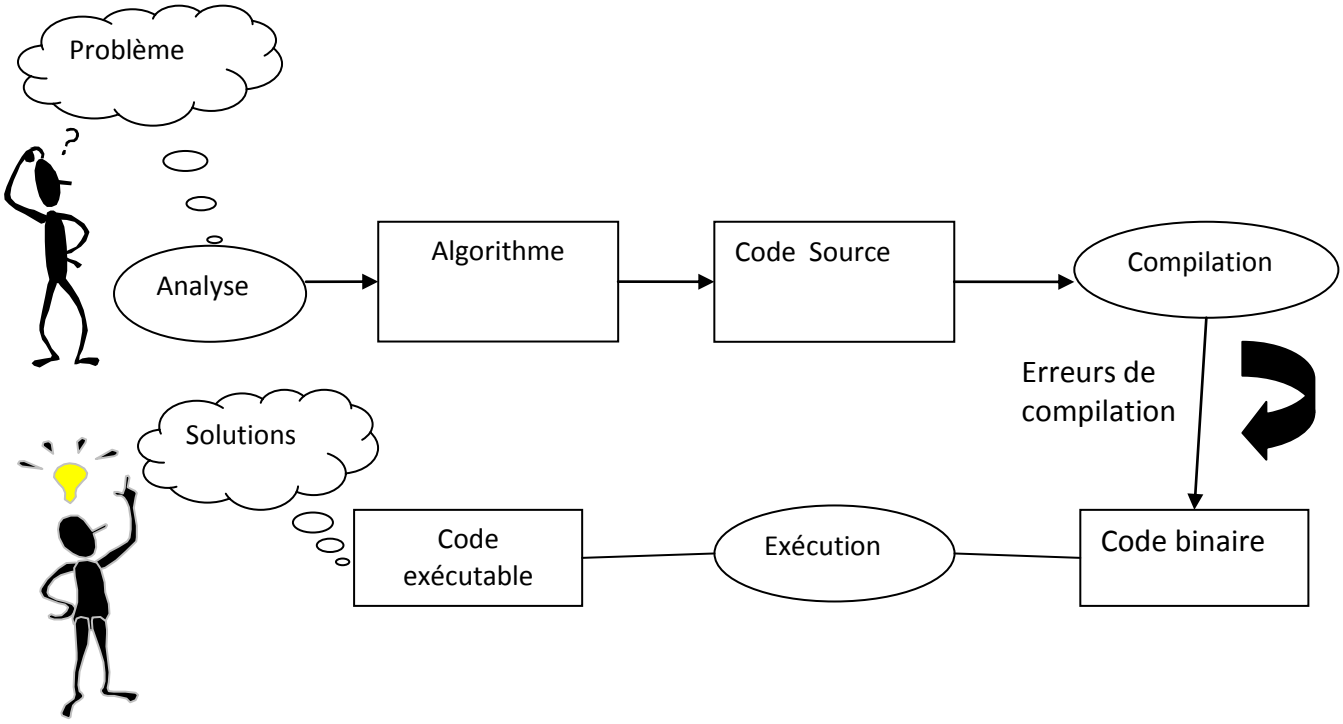


Figure 1 : Les niveaux de programmation

4. Les étapes de programmation



5. Les objets manipules dans un algorithme

- Objets simples et structurés
- Trois types d'objets simples : les constantes, les variables, et les expressions
- Chaque objet est caractérisé par :
  - L'identificateur : C'est le nom de l'objet
  - La valeur : C'est le contenu de l'objet
  - Le type : C'est la nature de l'objet

- **Les constantes**

Une constante est un objet qui a un nom fixe, un type fixe et une valeur fixe. **exemple** : pi=3.14

- **Les variables**

Une variable est un objet qui a un nom fixe, un type fixe, et une valeur qui change

- **Les expressions**

Une expression est un objet composé d'opérandes et d'opérateurs qui sont évalues pour donner un résultat.

Les opérateurs peuvent être : **Arithmétiques** (+ - / * DIV MOD) ,**Logiques** (ET OU NON), **Relationnels** (≠ ≤ < ≥ > =)

Les opérandes peuvent être : **Variables , Constantes, Expressions**

6. Les actions de base

- **L'affectation**

C'est une action qui consiste à affecter une valeur à une variable, elle est représentée par une ←

NB : Le type de la valeur doit être du même type de la variable

- **La Lecture**

Cette fonction permet de lire une valeur à partir d'un périphérique d'entrée (généralement le clavier), et stocker cette valeur dans une variable. Syntaxe : **lire(variable)**

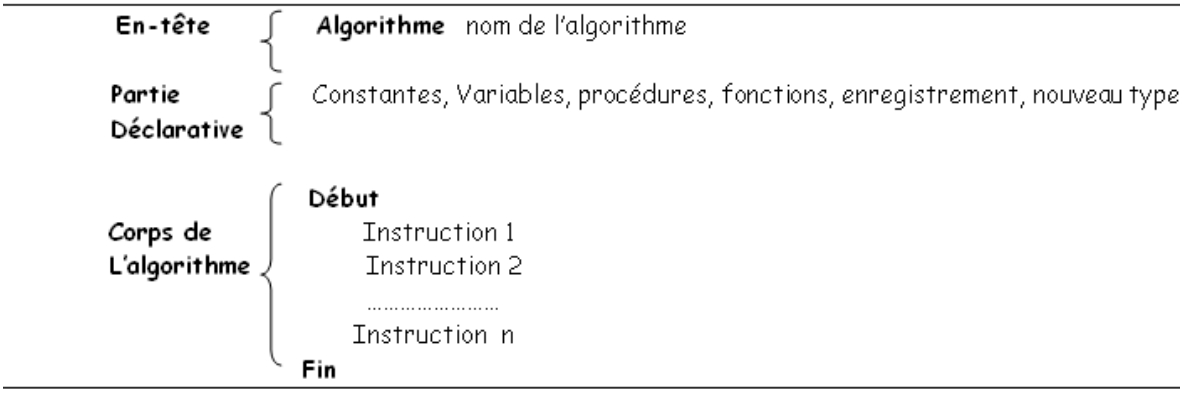
NB : Avec l'opération de lecture il faut respecter la compatibilité des types.

- **L'écriture**

Cette fonction affiche les résultats sur un périphérique de sortie (généralement l'écran) Syntaxe : **Ecrire (objet)**

- Une constante est affichée telle qu'elle est
- Une variable est remplacée par sa valeur
- Une expression est évaluée et le résultat est affiché

7. La structure d'un algorithme



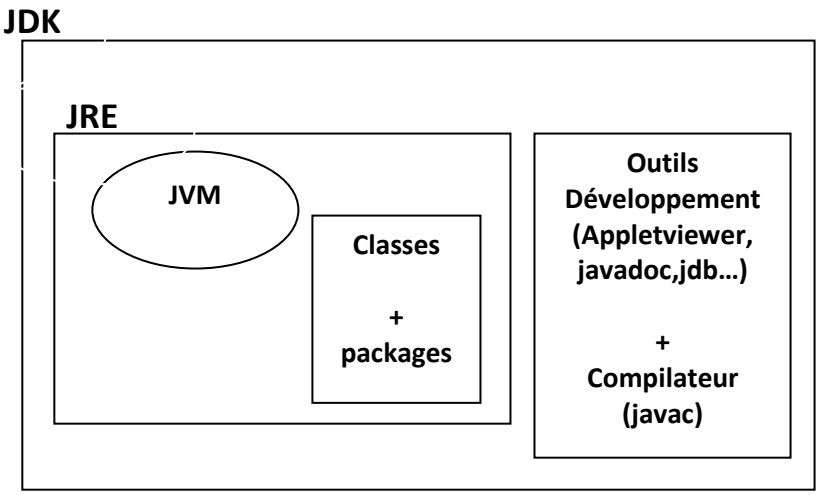
Introduction :

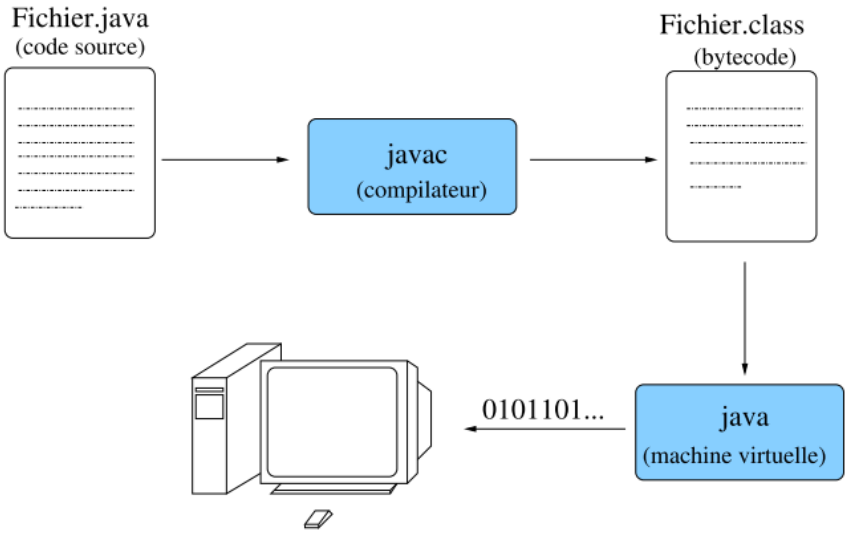
- Java est un langage de programmation orienté objet
- développé par SUN microsystems en 1995 (Sun est racheté par Oracle Corporation en 2009)
- Origine du nom : **J**ames Gosling,**A**rthur **V**an Hoff, and **A**ndy Bechtolsheim
- site officiel www.oracle.com
- dernière version stable : JDK 1.8 (ou Java 8)
- Présent dans de très nombreux domaines d'application : ordinateurs, console de jeux, des serveurs d'applications aux téléphone portables et cartes à puces, navigateur web
  « 3 billion devices run Java »

1. **Les plateformes Java :**
   - **Java EE** : "Enterprise Edition". Rajoute certaines fonctionnalités pour les entreprises (applications réparties)
   - **Java ME** : "Micro Edition" Edition qui sert à écrire des applications embarquées Ex. : téléphone portable, carte à puce
   - **Java SE** : "Standard Edition" : destinée typiquement aux applications pour poste de travail
   - **JavaFX** : outil de création des interfaces graphiques pour toutes sortes d'application java (applications mobiles, applications sur poste de travail, applications Web...) (En fait, JavaFX est une évolution de Java, elle permet de créer des RIA (Rich Internet Applications), c'est-à-dire des applications contenant des vidéos, de la musique, des effets graphiques très intéressants)

Toutes les plateformes contenant :

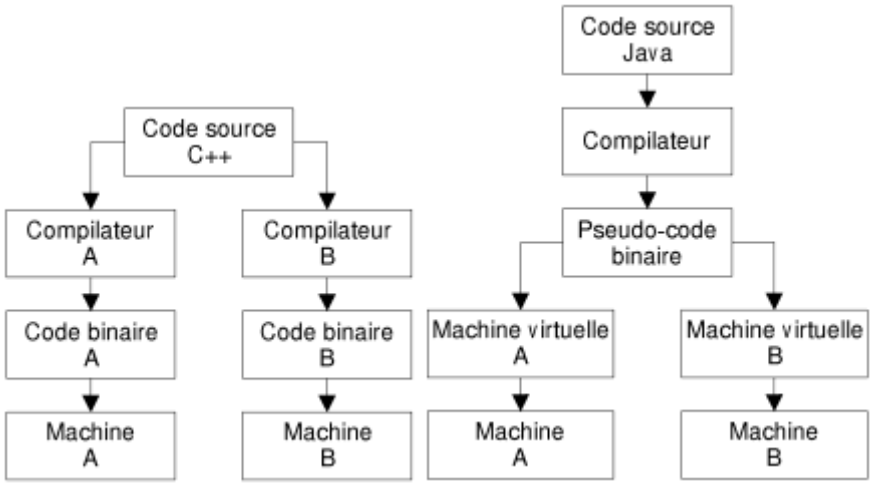
JRE(*Java Runtime Environment*) et **JDK** (*Java Development Kit*)





Principe d'exécution d'un programme java

- JVM : ("Java Virtual Machine") le logiciel qui interprète le ***bytecode*** généré par le compilateur Java
- Java est portable, c'est-à-dire qu'il ne dépend pas d'une plate-forme donnée



Comparaison java/C++

NB : Le code intermédiaire produit « ByteCode » est indépendant des plates-formes, il pourra être exécuté sur tous types de machines et systèmes à condition qu'ils possèdent l'interpréteur du code java **(JVM)**

2. Les environnements de développement java :

Jcreator Jedit Netbeans Eclipse …..

3. Les avantages de java

- Le Langage java peut générer des applications, des applets, des servelets, etc.
- Java est un langage simple, orienté objet, distribué, robuste, sûr, indépendant des architectures matérielles, portable, de haute performance, multithread (Une applications peut être décomposée en unités d'exécution fonctionnant simultanément)
- Ecrire une fois, exécuter partout (portable)
- Gestion automatique de la mémoire (garbage collector)

4. Premier programme java

```
class Premierprogramme {
public static void main(String[ ] args) {
System.out.println("Hello World!");
}
}
```

- Nom du programme : ***PremierProgramm.java***
- Par convention le nom d'une classe JAVA commence par une Majuscule. N'utiliser que les lettres **[a-z] et [A-Z] et [0-9]**
- En Java, les majuscules et les minuscules sont considérés comme des caractères différents
- **{ }** début et fin d'un bloc d'instructions
- Le mot **main** indique que cette méthode est la méthode principale de la classe
- le mot **void** signifie que la méthode main ne renvoie aucune valeur.
- les mots **public** et **static** décrivent chacun une caractéristique de la méthode

  (**public** : méthode visible, **static** : spécifique à la classe et non aux objets)
- **args[ ]** est le paramètre d'entrée de type String de la méthode main.
- ***System.out.println*** *:* fonction d'affichage avec saut de ligne
- ***System.out.print*** *:* fonction d'affichage su la même ligne

5. Types primitifs en java

boolean (true/false)

char (16bits) : les caractères en UNICODE

Types entiers

byte : (1 octet) entiers compris entre -128 et +127 (-2^7 et 2^7-1)
short : (2 octets) entiers compris entre -32768 et +32767 (-2^{15} et $2^{15}-1$)
int : (4 octets) entiers compris entre -2147483648 et +2147483647 (-2^{31} et $2^{31}-1$)
long : (8 octets) entiers compris entre -9223372036854775808 et +9223372036854775807 (-2^{63} et $2^{63}-1$)

Types réels

Float (32 bits ieee754 sp)
double (64 bits ieee754 dp)

Chaînes de caractères

- Les chaînes de carac ne correspondent pas à un type de données mais à une classe
- Une chaîne de caractère est un objet possédant des attributs et des méthodes.

```
String        s = "Ahmed Mohamed";
```

6. Déclaration et initialisation

- Les constantes

```
final int n = 20 ;
final char c='c';
final float x=15.25f;
final float y=15.25d;
final String s = "Ahmed Mohamed";
```

- Les variables

```
int n = 20 ;
char c='c';
float x=15.25f;
float y=15.25d;
String s = "Ahmed Mohamed";
```

7. Conversion de type de données (transtypage)

a. conversion implicite

- une conversion implicite consiste en une modification du type de donnée effectuée automatiquement par le compilateur
- le compilateur ne retournera pas d'erreur

Exemple :

```
byte a=15 ;  int b=a ;
int  a=15 ;   float b=a ;
```

b. conversion explicite

- une conversion explicite (casting) consiste en une modification du type de donnée forcée

```
-     int  a=15 ;  short b=(short)a ;
- float a=-9.25f;
   int b=(int) a;
a=-9.25 et b=-9
```

byte → short→int →long→float→double

↑

char

Exemple :

```
char c='A'; int x=c;    affiche x =65
char c='A'; short x=c;   faux
char c='A'; short x=(short)c; juste
```

8. Les opérateurs en java

Arithmétiques + - * / (entire ou réelle) % (reste d'une division entière ou réelle)

Affectation =

Assignation += *= /= %= -=

Incrémentation ++ décrémentation - -

Relationnels < <= > >= == !=

Logiques && || !

Exemple :

```
int a=5;
float x=a%2.3f;
System.out.println(x);            //affiche 0.40
```

9. Les commentaires en java

```
// Un commentaire sur une ligne
/*  sur plusieurs */
```

10. Les structures conditionnelles en java

a. Instruction if

- la condition doit être entre des parenthèses
- condition composée (utiliser les opérateurs && ||)

```
if (condition)
{
bloc1
}
else
{
bloc 2
}
```

b. Instruction switch

```
switch (variable) {

case valeur1 :
instructions
break;

case valeur2 :
instructions
break;

case valeurN :
instructions
break;

default:  instructions
}
```

11. Les boucles en java

i. FOR

```
for (int i=1; i<6; i++) {
        System.out.println(i);
      }
```

ii. WHILE

```
while (condition )
{
bloc d'instructions
}
```

iii. DO WHILE

```
do {
bloc d'instructions
} while (condition );
```

12. fonction de lecture

```
import java.util.Scanner;
//Ceci importe la classe Scanner du package java.util
import java.util.*;
//Ceci importe toutes les classes du package java.util

Scanner sc = new Scanner(System.in);
String str = sc.nextLine();
int str = sc.nextInt();
float x = sc.nextfloat();
double d = sc.nextDouble();
long l = sc.nextLong();
byte b = sc.nextByte();
```

Pour lire un caractère :

```
Scanner sc = new Scanner(System.in);
char carac= sc.next().charAt(0);
```

NB : pour changer la langue il faut suivre ces étapes :
Window > Preferences > Java > Installed JREs > select preferred JRE > Edit and then add the following to *Default VM arguments*: **-Duser.language=en -Duser.country=US**

13. Les tableaux en java

a. Avec initialisation

Type nom []:{contenu};

Exemple :

```
int tableau[] = {0,1,2,3,4,5,6,7,8,9};
double tableauDouble[] = {0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0};
char tableauCaractere[] = {'a','b','c','d','e','f','g'};
String tableauChaine[] = {"chaine1","chaine2","chaine3" , "chaine4"};
```

b. Déclaration (Avec une taille définie)

```
Scanner sc = new Scanner(System.in);
int taille=sc.nextInt();
float tableau[] = new float[taille];
```

c. Lire et afficher

```
Scanner sc = new Scanner(System.in);
float tableau[] = new float[6];

for (int i=0; i<tableau.length; i++) {
tableau[i]=sc.nextFloat();

}
for (int i=0; i<tableau.length; i++) {
System.out.print(tableau[i]+"   ");

}
```

14. Les matrices en java

a. Avec initialisation

```
int matrice[][] = { {1,2,3,4,5},{6,7,8,9,10} };
```

b. Déclaration (Avec une taille définie)

```
Scanner sc = new Scanner(System.in);
int lignes=sc.nextInt();
int colonnes=sc.nextInt();
int matrice[][]=new int [lignes][ colonnes];
```

c. Lire et afficher

```
int matrice[][] = new int [3][5];

//lecture
for(int i = 0; i < matrice.length; i++)

for(int j = 0; j < matrice[i].length; j++)

matrice[i][j]=sc.nextInt();

//affichage
for(int i = 0; i < matrice.length; i++)
{
for(int j = 0; j < matrice[i].length; j++)
{
System.out.print(matrice[i][j]+"  ");
}
System.out.println();
}
```

NB :

- Les indices commencent par zéro
- Les tableaux sont initialisés par 0 lors de la déclaration
- On peut utiliser l'affectation pour les tableaux

```
int tableau1[] = {0,1,2,3,4,5,6,7,8,9};

int tableau2[]=tableau1;
```

Attention : tableau1 et tableau2 font référence au même tableau

- pour trier un tableau :

```
import java.util.Arrays;

public class tp1 {

public static void main(String[] args) {

int tableau[] = {0,1,2,3,-7,4,5,6,7,8,9};
Arrays.sort(tableau);
for (int i=0; i<tableau.length; i++) {
System.out.print(tableau[i]+"   ");
```

- pour copier un tableau

```
float tableau1[] = {0,1,2,3,4,5,6,7,8,9};
float tableau2[] = {20,11,-3,17};

System.arraycopy(tableau1, 4, tableau2, 1, 2);
// 0 :case départ tableau1
//0 : case départ tableau2
//10: nombre d'éléments

for (int i=0; i<tableau2.length; i++) {
System.out.print(tableau2[i]+"   ");
//tableau2 devient: 20.0   4.0   5.0   17.0
}
```

15. Fonctions mathématiques en Java

- Elles sont fournies par la classe Math.
- Les angles sont toujours exprimés en radians.

Exemple :

```
double x=Math.E; double x=Math.PI;
```

Nom fonction	Rôle	En-têtes
atan2	Arc tangente (a/b) (angle dans l'intervalle [-pi/2, pi/2])	double atan2 (double a, double b)
ceil	Arrondi à l'entier supérieur	double ceil (double a)
cos	Cosinus	double cos (double a)
exp	Exponentielle	double exp (double a)
floor	Arrondi à l'entier inférieur	double floor (double a)
IEEEremainder	Reste de la division de x par y	double IEEEremainder (double x, double y)
log	Logarithme naturel (népérien)	double log (double a)
max	Maximum de deux valeurs	double max (double a, double b) float max (float a, float b) int max (int a, int b) long max (long a, long b)
min	Minimum de deux valeurs	double min (double a, double b) float min (float a, float b) int min (int a, int b) long min (long a, long b)

Nom fonction	Rôle	En-têtes
abs	Valeur absolue	double abs (double a) float abs (float a) int abs (int a) long abs (long a)
acos	Arc cosinus (angle dans l'intervalle [-1, 1])	double acos (double a)
asin	Arc sinus (angle dans l'intervalle [-1, 1])	double asin (double a)
atan	Arc tangente (angle dans l'intervalle [-pi/2, pi/2])	double atan (double a)
pow	Puissance (a^b)	double pow (double a, double b)
random	Nombre aléatoire dans l'intervalle [0, 1[	double random ()
rint	Arrondi à l'entier le plus proche	double rint (double a)
round	Arrondi à l'entier le plus proche	long round (double a) int round (float a)
sin	Sinus	double sin (double a)
sqrt	Racine carrée	double sqrt (double a)
tan	Tangente	double tan (double a)
toDegrees	Conversion de radians en degrés	double toDegrees (double aRad)
toRadians	Conversion de degrés en radians	double toRadians (double aDeg)

16. Les méthodes en Java (statiques)

- Une fonction (méthode) un sous-programme qui permet d'effectuer un ensemble d'instructions par simple appel de la fonction dans le corps du programme principal
- une simplicité du code et donc une taille de programme minimale
- une fonction peut faire appel à elle-même, on parle alors de fonction récursive

a. Déclaration

<Mode d'accès > static <type de retour> <nom de la fonction> (<listes des paramètres avec types>)
{ déclarations de variables (locales à cette méthode) ;
Instructions ;
Return résultat ;
}

- <Mode d'accès > : public, private, protected (facultatif) (on verra ces modes en détail)
- <type de retour> : peut-être un type primitif, un objet, ou alors le mot-clé ***void*** si la méthode ne retourne aucune valeur
- Le nom de la méthode doit commencer par une minuscule (par convention)
- La liste des paramètres contient : nom de chaque paramètre avec son type séparé par des virgules
- On peut avoir des méthodes sans paramètres (affichage)
- Les variables locales sont utilisables dans toute la classe dans laquelle elles sont déclarées
- **Static** est utilisé pour les variables globales (une variable globale est utilisable dans main et dans le reste des méthodes)

- Les méthodes sont déclarées après ou avant main
- Si la méthode renvoie une valeur, elle sera appelée dans (une affectation, expression, affichage) sinon elle sera appelée par son ***nom(liste d'arguments) ;***
- On peut appeler une méthode d'une classe exemple : double x= Math.pow(5,2) ;

b. Passages des paramètres

- Le mode de passage des paramètres (types simples) en Java est toujours le passage par valeur.
- Lorsqu'on fait appel à une méthode, Java passe une copie des valeurs des variables à la méthode. C'est-à-dire que la valeur du paramètre réel utilisé lors de l'appel est copiée dans la valeur du paramètre formel de la méthode. Si la méthode modifie la valeur de la variable, cela n'affectera en rien le paramètre réel étant donné que c'est une copie dont dispose la méthode.

Exemple

```
static void modifier(float x)
{
x=x*2;
}
public static void main(String[] args) {

float a=15.25f;
System.out.println(a);
modifier(a);
System.out.println(a);

      }

      }
```

Le programme affiche 15.25 15.25

- Les tableaux et les objets sont passés par référence (pas de copie, donc ils seront modifiés) en d'autres termes : la fonction reçoit les adresses de ces variables.

Exemple

```
      static void modifier(int tableau[])
      {
      for (int i=0; i<tableau.length; i++)
      if (tableau[i]>=5)
      tableau[i]=-1;
      }
      public static void main(String[] args) {

      int tableau[] = {0,1,2,3,4,5,6,7,8,9};
      for (int i=0; i<tableau.length; i++) {
      System.out.print(tableau[i]+"   ");

      }
      modifier(tableau);
      System.out.println();
for (int i=0; i<tableau.length; i++) {
System.out.print(tableau[i]+"   "); }}}
```

c. La surcharge (overloading)

- C'est la possibilité de déclarer plusieurs méthodes avec le même nom, à condition que leurs arguments diffèrent (en type et/ou en nombre)

Exemple :

```
static int somme(int p1, int p2) {
return p1+p2;
}
static int somme(int p1, int p2, int p3) {
return p1+p2+p3;
}
static double somme(double p1,double p2){
return p1+p2;
}

public static void main(String[] args) {
int a=5,b=6,c=18;
float x=10, y=2.5f;

System.out.println(somme(a,b));
System.out.println(somme(a,b,c));
System.out.println(somme(x,y));
}

}
```

Affichage

```
11
29
12.5
```

- Java choisit la méthode correspondante selon le type et le nombre de paramètres
- Si on déclare `static float somme(int p1, int p2, int p3)` ***Java signale une erreur***

static

- attribut static = variable de classe = existe en un seul exemplaire quelque soit le nombre d'instanciation (ou sans instanciation)
- méthode static = méthode de classe = méthode "sans envoi de message" appliquée à la classe et non à un objet.
- code static = static { instructions } initialise les variables static d'une classe
- Le mot clé static permet alors à une méthode de s'exécuter sans avoir à instancier la classe qui la contient

Chapitre2 : la programmation orientée objet (POO)

1. Paradigme de programmation

« Un paradigme de programmation est une manière de penser, d'écrire et de structurer un programme dans un langage de programmation ». Il existe de nombreux paradigmes :

- **Programmation impérative (la plus courante)** : un programme est vu comme une suite ordonnée de transformations d'états (Fortran, Pascal, c, java,…….)
- **Programmation fonctionnelle :** un programme est vu comme une suite d'évaluations de fonctions (Ml,Lisp,Haskell,Ocaml….)
- **Programmation orientée objet :** le programme est vu comme une collection d'objets en interaction (C++,C#, Java, Python…. Swift(appel2014))
- **Programmation logique :** exprimer les problèmes et les algorithmes sous forme de prédicats (Prolog…)
- **Programmation par contraintes** : résoudre des problèmes combinatoires de grandes tailles tels que les problèmes de planification et d'ordonnancement (python-constraint,….)
- **Programmation concurrente :** l'exécution en parallèle de plusieurs processus
- **La programmation procédurale :** Un programme est composé de plusieurs procédures, avec la possibilité de réutiliser le même code à différents emplacements dans le programme

2. Programmation orientée objet (POO)

- En P.O.O, un programme met en œuvre différents objets. Chaque objet associe des données et des méthodes agissant sur des données
- Alan Kay, dans les années 70'
- Quelques langages objets : C++, Java, Ada, PHP, Python, ...
- Smalltalk (Alan Kay 1972). Objective C (début des années 1980), C++ (C with classes) en 1983 JAVA (1995 SUN microsystems)

2.1 Notion d'objet

- Un objet est un concept, une idée ou une entité du monde physique par exemple : voiture, personne, étudiant, animal, stylo ……
- Un objet est caractérisé par trois notions :
    1. **Une identité :** qui permet de le distinguer de manière unique des autres objets
    2. **Les attributs** : les données de l'objet / les variables qu'il contient et représentant son état
    3. **Les méthodes (**comportement) C'est l'ensemble des actions que l'objet peut réaliser

Exemple : objet voiture---- attribut : contenu réservoir =40 litres après un parcours de 100km ------ contenu réservoir =20 litres

NB :

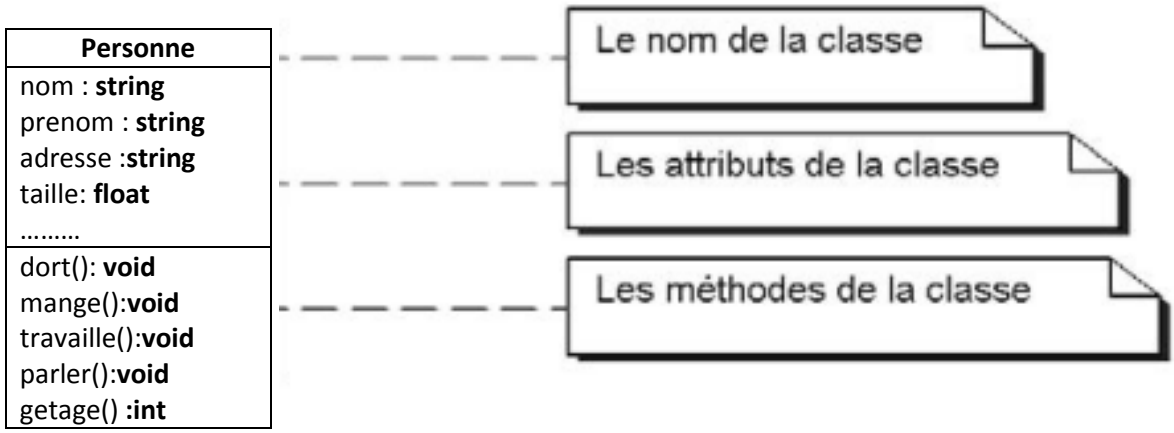
- **les attributs** d'un objet sont stockés dans des variables et ses **méthodes** sont implémentés à l'aide de (procédures / fonctions).
- En POO, un programme est constitué d'un ensemble d'objets chacun disposant d'une partie procédures (méthodes) et d'une partie données (attributs). Les objets interagissent par envoie/réception de messages.
- Un objet est une instance de classe, c'est-à-dire un exemplaire utilisable crée à partir de cette classe

2.2 Notion d'abstraction

- C'est l'identification et le regroupement des caractéristiques et des traitements communs applicables à des éléments variés
- Les regroupements se font dans des entités appelées **Classes**

2.3 Notion de classe

- Une classe encapsule, c'est-à-dire regroupe des propriétés (les attributs) et des comportements (les méthodes)
- les membres d'une classe = propriétés + comportements
- Une classe est un "moule" pour fabriquer des objets
- Par exemple, la classe Personne définit des propriétés (nom, prenom, adresse, date de naissance,taille,….) et des comportements (travailler, manger, dormir…..)



NB :

- Un objet est une instance de classe, c'est-à-dire un exemplaire utilisable crée à partir de cette classe
- l'instanciation d'un objet : c'est à dire la création d'un espace en mémoire pour cet objet
- Le constructeur est une méthode qui est appelée lors de l'instanciation d'un objet

Exemple : Instanciation de 2 objets de la classe Personne

P1
nom : **Bouchareb** prénom : **Mohamed** adresse :**02 rue de la liberté Sétif** taille: **1.68** ……
dort(): **void** mange():**void** travaille():**void** parler():**void** getage() **:int**

P2
nom : **Noureddine** prénom : **Samir** adresse :**54 rue de l'indépendance Alger** taille: **1.90** ……
dort(): **void** mange():**void** travaille():**void** parler():**void** getage() **:int**

4. Constructeur et initialisation des objets :

- Pour créer un objet (instancier) à partir d'une classe, on utilise l'opérateur **new**
- **new** fait appel automatiquement à une méthode spéciale appelée : le constructeur
- le constructeur permet d'initialiser les données membres d'un objet
- il porte le même nom de la classe dans laquelle il est défini
- un constructeur n'a pas de type de retour (même pas **void**)
- la définition d'un constructeur n'est pas obligatoire
- on peut surcharger plusieurs constructeurs pour la même classe

4.1 constructeur sans arguments

```
public class Personne {
 String nom;
 int age;

public void afficherage()
{
System.out.println("AGE:"+age);
}

}
public class Test {

public static void main(String[] args) {

   Personne a=new Personne ();
         a.nom="semchedine";
         a.age=32;

         a.afficherage();

   }
}
```

```
import java.util.Scanner;
public class Test {

public static void main(String[]
args) {
Scanner sc=new Scanner(System.in);
      Personne a=new Personne ();
            a.nom=sc.nextLine();
            a.age=sc.nextInt();
            a.afficherage();


      }
      }
```

4.2 constructeur avec arguments

```
public class Personne {
 String nom;
 int age;
public Personne (String Nom)
{
nom=Nom;
}

public void afficherage()
{
System.out.println("AGE:"+age);
}

}
public class Test {

public static void main(String[] args) {

   Personne a=new Personne (“Mohamed Amine”,25);

         a.afficherage();

   }
}
```

4.3 constructeurs multiples

```
public class Personne {
 String nom;
 int age;
public Personne ()
{
}

public Personne (String Nom)
{
nom=Nom;
}

public Personne (String Nom,int Age)
{
nom=Nom;
age=Age;
}

public void afficherage()
{
System.out.println("AGE:"+age);
}

}
public class Test {

public static void main(String[] args) {
Personne a=new Personne();
Personne b=new Personne (“Bouchareb Amine”);
Personne c=new Personne (“Mohamed mustafa”,25);


   }
}
```

NB : Si aucun constricteur n'est défini explicitement, le compilateur JAVA crée un par défaut et initialise tous les attributs par la valeur nulle (o pur les variables numériques et NULL pour les références)

Attention :

```
Personne a,b ;
a=new Personne();
a.nom="Mohamed mustafa"
b=a;
b.nom="Fadila Lafri"
System.out.println(a.nom);
```

5. La notion d'accessibilité (L'encapsulation)

- Le concept d'accessibilité (encapsulation) définit la possibilité qu'a le concepteur d'une classe de limiter l'accès à certains éléments (les classes, les attributs et les méthodes)

Java définit quatre niveaux d'accès pour les membres de la classe (attributs+ méthodes) :

- **public** : les membres de la classe sont accessibles de n'importe quelle méthode de n'importe quelle classe
- **protected** : les membres de la classe sont accessibles seulement par ses classes filles et par les classes du même package
- **« aucun mode » (par défaut) :** les membres de la classe sont accessibles seulement par les classes du même package
- **private** : les membres de la classe sont accessibles seulement par les méthodes de la même classe

  public →protected→ « aucun mode »→private

Encapsulation pourquoi faire?

- Ne doit être visible de l'extérieur que ce qui est nécessaire, les détails d'implémentation sont « cachés »
- L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet
- En interdisant l'utilisateur d'agir directement sur les attributs d'un objet, qu'avec les méthodes de cet objet, on est capable de s'assurer de l'intégrité des données (on pourra par exemple s'assurer que le type des données fournies est compatible, ou encore que les données se trouvent bien dans l'intervalle attendu)
- Toute modification d'un attribut de l'objet est maîtrisée

Remarque :

- Il est recommandé de déclarer les attributs en mode private, pour les protéger d'une mauvaise utilisation par le programmeur

- On crée des **accesseurs** et **mutateurs** (méthodes getters et setters) pour permettre une modification sûre des variables d'instance (les attributs)

6. Accesseurs et mutateurs

6.1 Accesseurs (getters)

- Un accesseur est une méthode permettant de récupérer le contenu d'un attribut protégé. Son type de retour est le type de la variable à renvoyer
- Par convention de nommage on commence le nom de l'accesseur par le préfixe ***get***

6.2 Mutateurs (setters)

- Un mutateur est une méthode permettant de modifier le contenu d'un attribut protégé. Ne possède pas de type (**void**)
- doit avoir comme paramètre la valeur à assigner à l'attribut. Le paramètre doit donc être du même type de l'attribut
- Par convention de nommage on commence le nom du mutateur par le préfixe ***set***

NB : Grâce aux accesseurs, vous pourrez afficher les variables de vos objets, et grâce aux mutateurs, vous pourrez les modifier

Exercice : Ecrire une classe **Etudiant** comportant les attributs : nom, age, moyenne

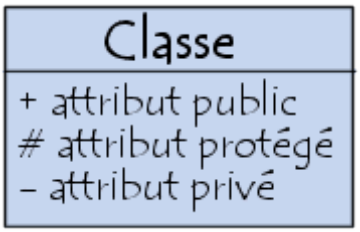
a) Ajouter une méthode **affichertout**() : qui permet d'afficher toutes les information

b) Créer une classe **Test** contenant la méthode **main()** : permettant (1) d'instancier deux objets **Etudiant** , (2) saisir les variables de chaque instance (par des valeurs de votre choix)puis (3) invoquer la méthode **afficherTout()** de chaque instance

c) Supposant que les attributs sont en niveau **private**, Ajouter les méthodes getters/setters et effectuer les changements dans **Test**

d) Modifier **setmoyenne()** pour affecter des moyennes valides

```
public class Etudiant {
 private String nom;
 private int age;
 private float moyenne;

 public String getnom()
 {
    return nom;
 }
 public void setnom(String Nom)
 {
    nom=Nom;
 }
 public int getage()
 {
    return age;
 }
 public void setage(int Age)
 {
    age=Age;
 }
 public float getmoyenne()
 {
    return moyenne;
 }
 public void setmoyenne(float
Moyenne)
 { if (Moyenne>=0 &&
Moyenne<=20)
    moyenne=Moyenne;
 }
public void affichertout()
{
   System.out.println("NOM:"+get
nom()+"\n"+"AGE:"+getage()+"\n"+
"Moyenne:"+getmoyenne());
   }

}
```



- Les langages multi-paradigmes

Remarque : L'encapsulation permet d'assurer la cohérence des valeurs des attributs, par exemple en ne permettant de modifier deux attributs qu'ensemble car leurs valeurs sont liées.

Exemple :

```
public class abonné  {
      public float credit;
      public float flexy;
      public void affichercrédit()
       {
System.out.println("votre crédit="+credit);

       }

   }
```

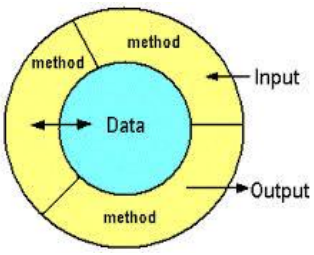
```
public class Abonné  {

      private float credit;
      private float flexy;

      public void setFlexy (float Flexy)
       {
            flexy=Flexy;
            credit=credit +flexy;
       }

      public float affichercrédit()
       {
           System.out.println("votre crédit="+credit);

       }

   }
```

```
public class Test {

public static void
main(String[] args) {
Abonné a1=new Abonné ();
a1.flexy=500 ;
a1. affichercrédit() ;

      }
```

```
public class Test {

public static void
main(String[] args) {
Abonné a1=new Abonné ();
a1.setFlexy(500) ;
a1. affichercrédit() ;

      }
```

Exercice : soit les projets suivants(voir l'accessibilité de chaque attibut/méthode)
Projet1 (classe Personne{private String nom, private int age , public afficherage()} , classe 1, classe2, classe 3, …..)
Projet2(classe Etudiant , classe 1, classe2, classe 3, …..)
Projet3(classe Enseignant , classe 1, classe2, classe 3, …..)

Schéma d'encapsulation



7. Les membres de la classe et les membres d'instance (objet)

- Les membres statiques d'une classe sont précédés par le mot-clé ***static***
- les membres statiques appartenant à la classe et les membres d'instance aux objets
- Un membre statique identifie exactement un seul emplacement de stockage. Quel que soit le nombre d'objets créés
- *Le membre statique peut être appelé sur une classe même quand aucune instance de la classe n'a été créée*
- *Les méthodes statiques ne peuvent pas accéder à des membres non statiques*
- Pour accéder à un membre **static**, on utilise le nom de la classe suivi d'un point puis du nom du membre (attribut/méthode)
- *Ça fonctionne aussi en utilisant le nom de l'objet suivi d'un point puis de l'attribut*
- Exemple d'utilisation des attributs statiques
    1- conserver le nombre d'objets ayant été créés
    2- stocker une valeur qui doit être partagée par tous les objets

Exercice : on suppose que la classe **Etudiant** possède un attribut **numéro.**
On veut attribuer automatiquement un numéro séquentiel lors de la création de chaque objet étudiant

Questions :

- écrire la nouvelle classe étudiant
- Ecrire un constructeur Etudiant (String Nom, int Age)
- Ecrire une classe **Test.java**

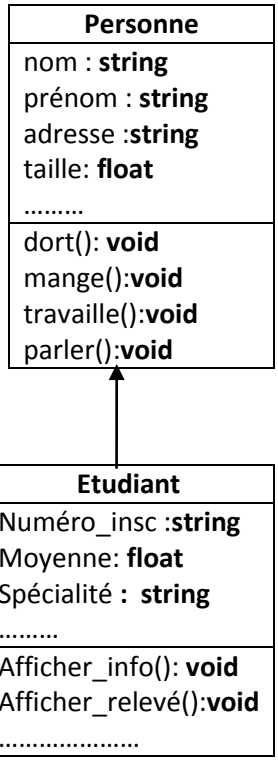
```
public class Etudiant {
      private static int nombre=0;
      private    int numéro;
      private     String nom;
      private      int age;
 public Etudiant(String Nom, int
Age)
 {
       nom=Nom;
       age=Age;
      numéro= ++nombre;
 }
 public String getnom()
 {
       return nom;
 }
 public void setnom(String Nom)
 {
       nom=Nom;
 }
 public int getage()
 {
       return age;
 }
 public void setage(int Age)
 {
       age=Age;
 }
 public void affichertout()
 {
       System.out.println("NOM:"+get
nom()+" "+"AGE:"+getage()+"
"+"Numéro:"+"Numéro:"+numéro);
       }

public class Exo2 {

public static void main(String[]
args) {
Scanner sc=new Scanner(System.in);
System.out.println("Entrez svp les
infrmations des étudiants");
Etudiant
et1=new Etudiant("touhami
mohamed",25),
    et2=new Etudiant("Dib
Farid",19),
    et3=new Etudiant("Kharchi
Ali",23);
et1.affichertout();
et2.affichertout();
et3.affichertout();

System.out.println(Etudiant.nombre)
;
       }
}
```

8. Notion d'héritage (Inheritance)

- Problème : on veut informatiser la gestion du personnel (étudiant, enseignant, administrateur, employé…..). On remarque qu'il y a des répétitions. Afin d'éviter la répétition des éléments constituant chacune des classes, il est préférable de factoriser toutes ces caractéristiques communes pour en faire une nouvelle classe plus généraliste.

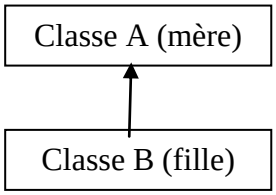
- L'héritage (pouvant parfois être appelé dérivation de classe) est une relation entre différentes classes permettant de définir une nouvelle classe en se basant sur les classes existantes.
- Il a pour objectif de hiérarchiser les classes et les objets.
- La classe fille (sous-classe) contient les attributs et les méthodes de la classe mère (super-classe) plus de nouveaux attributs et méthodes.
- On peut généralement exprimer la relation qui lie une classe fille à sa mère par la phrase "est un" (de l'anglais "is a").

Exemple :
Un étudiant est une sous-classe de personne

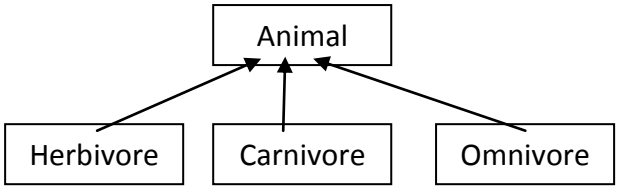


On distingue 2 types d'héritage :

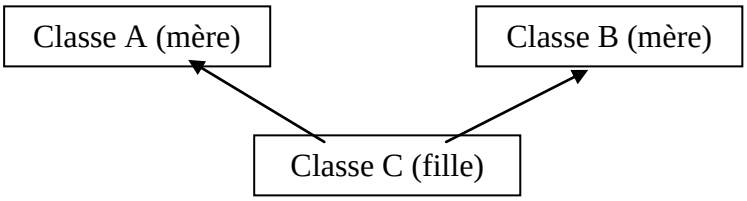
1. **Héritage simple :** une classe hérite d'une unique super-classe



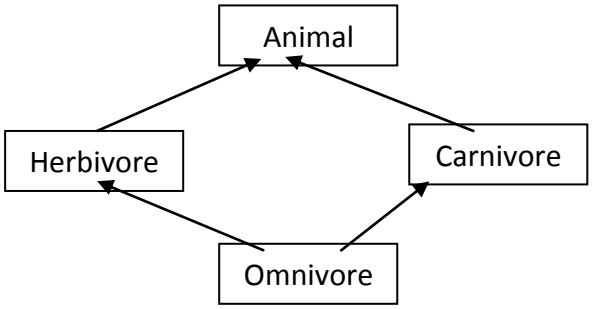
Exemple :



2. **Héritage multiple :** une classe peut hériter de plus d'une super-classe. Ainsi, cette technique permet de regrouper au sein d'une seule et même classe les attributs et les méthodes de plusieurs classes



Exemple :



Avantages de l'héritage :

- Une hiérarchie de classes facilite la solution de problèmes complexes
- Facilite la maintenance, le développement et les extensions (réutilisation de code).
- La modification de la superclasse implique la modification automatique de toutes les sous-classes.
- **généralisation** : factorisation de classes en regroupant des propriétés communes
- **spécialisation** : ajout d'attributs et méthodes

Remarques:

-Les attributs et les méthodes peuvent être hérités à un niveau **n**, c'est-à-dire qu'un objet peut utiliser une méthode de la mère de sa mère et ainsi de suite

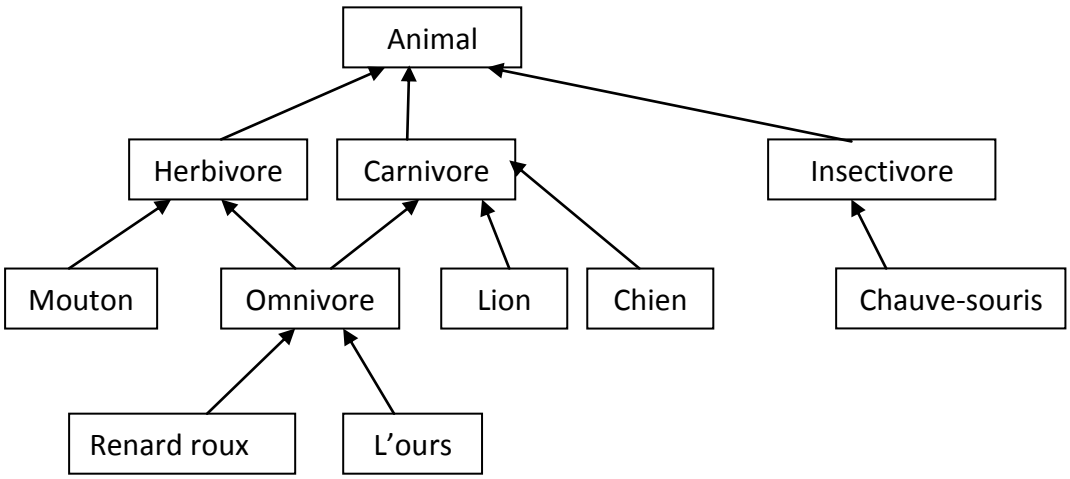
- Il faut toujours concevoir une hiérarchie de classe, c'est-à-dire organiser les classes de telles façon que les attributs et les méthodes communes à plusieurs classes soient placées dans une superclasse.

- Java ne possède pas de mécanisme d'héritage **multiple**

Exercice :

Placez dans des diagrammes de classes, les concepts suivants :

- humain, footballeur, avant-centre, sportif, skieur, Handballeur, Nageur
- Véhicule, véhicules sans moteur, véhicules avec moteur, vélo, voiture, camion
- Carnivore, Singe, caniche, Sloughi, Insectivore, Homme, Bulldog, Herbivore, Omnivore, Vache, Mouton, Lion, Chien, L'ours, Chauve-souris



Remarques :

- L'appel d'un constructeur de la classe mère doit être la première instruction du constructeur de la classe fille.
- Les membres (attributs et méthodes) déclarés privés dans une classe mère ne sont pas visibles dans une sous-classe. En d'autres termes, l'héritage n'implique pas la visibilité

- Une méthode de la classe mère peut être implémentée différemment dans une classe fille: la méthode est dite redéfinie
- Pour utiliser la méthode redéfinie de la classe mère on utilise le mot-clé **super** suivi du nom de la méthode

Exercice :

1) créer une classe **Employé** {nom, age, nb_enfants, getAge(), setAge()……. }

Et les sous-classe **Enseigant** { module, setModule(), getModule()……. Void AfficherToutEn()}

Administrateur { service, setService() ;getService()…….Void AfficherToutAd()}

2) Créer le constructeur **Employé**

3) On veut attribuer un numéro séquentiel pour chaque enseignant et un autre pour chaque administrateur. Créer les deux constructeurs **Enseigant Administrateur** (appeler le constructeur **Employé /**utiliser super())

4) Ajouter les méthodes Setters/Getters pour chaque classe *en vérifiant l'intégrité de l'âge et le nombre d'enfant*

5) créer une classe **Test.java** où vous instanciez un objet **en1** est un autre **ad1** par des exemple de votre choix / par des valeurs entrées au clavier et afficher toutes les informations des deux objets

6) changer le nombre d'enfant de **en1/** changer le module de **en1**

7) Modifier **Test.java** pour créer 5 objets enseignant et 3 Administrateurs.

8) Modifier le nom, le service de l'administrateur numéro 1.

```
public class employé {

	private String nom;
	private int age;
	private int nb_enf;
public employé(String Nom,int Age,
int Nb_enf)
	{
		nom=Nom;
		age=Age;
		nb_enf=Nb_enf;

	}
	public String getNom()
	{
		return nom;
	}
	public void setNom(String Nom)
	{
		nom=Nom;
	}
	public int getAge()
	{
		return age;
	}
	public void setAge(int Age)
	{if (Age>0)
		age=Age;
	}
	public int getNb()
	{
		return nb_enf;
	}
	public void setNb(int Nb)
	{if (Nb>0)
		nb_enf=Nb;
	}

}

public class Enseignant extends
employé{
	private static int nombre=0;
	private int numéro;
	private String module;

	public Enseignant(int Age, int
Nb_enf,String Nom,String Module)
	{
		Super(Nom,Age, Nb_enf) ;
		module=Module;
		numéro=++nombre;
	}
	public String  getModule()
	{
		return module;
	}

public void setModule(String Module)
	{
		module=Module;
	}

	public void affichertoutEn()
	{	System.out.print(
numéro+" "+getNom()+ " "+getAge()+ "
"+getNb +" "+getModule());
	}
```

```
}

public class Administrateur extends
employé {

      private static int nombre=0;
      private int numéro;
      private String service;

      public Administrateur(int Age,
int Nb,String Nom,String Service)
      {
            Super(Nom,Age, Nb_enf) ;
            service =Service;
            numéro=++nombre;
      }
      public String  getService()
      {
            return service;
      }
      public void setService(String
Service)
      {
            service=Service;
      }


      public void affichertoutAd()
      {     System.out.print(
numéro+" "+getNom()+ " "+getAge()+ "
"+getNb +" "+ getService());

      }
      }
}

public class Test {
      public static void
main(String[] args) {

Scanner sc1=new
Scanner(System.in),sc2=new
Scanner(System.in);
            //créer un tableau
contenant 3 objets
            Enseignant[] tab1 = new
Enseignant[3];
            //instancier les 3
objets
      System.out.println("entrez svp
les informations des 3
enseignants");

for (int i=0;i<tab1.length;i++)
tab1[i]=new
Enseignant(sc1.nextInt(),sc1.nextInt
(),sc2.nextLine(),sc2.nextLine());

for (int i=0;i<tab1.length;i++)

      tab1[i].affichertoutEn();

Administrateur[] tab2 = new
Administrateur[3];
      //instancier les 3 objets
      System.out.println("entrez svp
les informations des 3
administrateurs");

for (int i=0;i<tab2.length;i++)
            tab2[i]=new
Administrateur(sc1.nextInt(),sc1.nex
tInt(),sc2.nextLine(),sc2.nextLine()
);

for (int i=0;i<tab2.length;i++)

      tab2[i].affichertoutAd();

      }
}
```

9. Le polymorphisme

- Le nom de polymorphisme vient du grec et signifie qui peut prendre plusieurs formes
- C'est la possibilité de redéfinir une méthode dans des classes héritant d'une classe de base

Exemple :

On remarque que les informations des **Enseignants** et les **Administrateurs** ne sont pas affichées de la même façon

- **Solution 1** : écrire 2 méthodes **affichertoutEn**() et **affichertoutAd**()

Chaque fois qu'on a un objet, il faut regarder si c'est un **Enseignant** ou un **Administrateur** et appeler la bonne méthode pour afficher les informations. Ce n'est pas pratique !

- **Solution 2** : écrire la même méthode **affichertout** () dans les deux classes

Enseignant et les **Administrateur**

Quand vous appelez la méthode **affichertout** (on écrit **X. affichertout** ()) le système va regarder de quel objet il s'agit et appeler la bonne méthode **affichertout** (): celle de l'Enseignant si c'est un l'Enseignant ou celle de l'Administrateur si c'est un Administrateur

Exercice : L'exercice suivant explique le principe du **polymorphisme**

```
public class employé {

	private String nom;
	private int age;
	private int nb_enf;

	public employé(String Nom,int
Age, int Nb_enf)
	{
	  nom=Nom;
	  age=Age;
	  nb_enf=Nb_enf;
		}
	public String getNom()
	{
		return nom;
	}
	public void setNom(String Nom)
	{
		nom=Nom;
	}
	public int getAge()
	{
		return age;
	}
	public void setAge(int Age)
	{if (Age>0)
		age=Age;
	}
	public int getNb()
	{
		return nb_enf;
	}
	public void setNb(int Nb)
	{if (Nb>0)
		nb_enf=Nb;
	}
	}

public class Enseignant extends
employé{
	private static int nombre;
	private int numéro;
	private String module;

	public Enseignant(int Age, int
Nb,String Nom,String Module)
	{
		super(Nom,Age,Nb);
		setModule(Module);
		numéro=++nombre;
	}
	public String  getModule()
	{
		return module;
	}
	public void setModule(String
Module)
	{
		module=Module;
	}


	public void affichertout()
	{ System.out.print( numéro+"
"+getNom()+ " "+getAge()+ " "+getNb
+" "+getModule());
	}



}
public class Administrateur extends
employé {

	private static int nombre=0;
	private int numéro;
	private String service;

	public Administrateur(int Age,
int Nb,String Nom,String Service)
	{
		super(Nom,Age,Nb);
		setService(Service);
		numéro=++nombre;
	}
	public String  getService()
	{
		return service;
	}
	public void setService(String
Service)
	{
		service=Service;
	}


	public void affichertout()
	{	System.out.print(
numéro+" "+getNom()+ " "+getAge()+ "
"+getNb +" "+ getService());

	}

}

public class Test {
	public static void
main(String[] args) {
employé[] tab = new employé[4];
```

```
tab[0]=new
Enseignant(30,2,"AA","analyse");
tab[1]=new
Administrateur(50,4,"BB","comptabili
té");
tab[2]=new
Enseignant(45,3,"CC","informatique")
;
tab[3]=new
Administrateur(35,0,"DD","bibliothèq
ue");

for (int i=0;i<tab.length;i++)
      tab[i].affichertout();

      }
}
```

NB : C'est seulement à l'exécution que la machine virtuelle déterminera la méthode à invoquer selon le type effectif de l'objet référencé à ce moment là. Ce mécanisme s'appelle "Recherche dynamique de méthode" (Late Binding ou Dynamic Binding).

EXERCICES :

Solution (exo TD) :

1. C'est la méthode de Voiture car par défaut, la méthode exécutée est celle de la classe courante
2. Non car la classe Véhicule ne possède pas de méthode **klaxonner()**
3. Non, l'attribut est privé. Il faut le mettre soit **public** ou **protected**

Exercice 1(TP) :

Chaque étudiant possède 3 notes (math, info, anglais)

1. Ecrire la méthode **setNotes()** pour lire les 3 notes
2. Ecrire la méthode **getMoyenne()** qui renvoie la moyenne de chaque étudiant
3. Créer 5 objets **Etudiants**
4. Afficher la liste des étudiants admis (moyenne>=10)

Exercice 2 :

1. écrire la classe **Abonné**
2. Sachant que chaque abonné bénéficie d'un crédit initial de 100 DA. Ecrire le constructeur **Abonné (**`String nom`**)**
3. Ecrire la méthode **setFlexy()**
4. Ecrire la méthode **appeler()** qui permet de simuler l'appel téléphonique : on commence par lire le numéro à appeler, et on affiche le message « appuyer sur entrer pour terminer l'appel», puis on calcule la durée de l'appel en utilisant l'instruction

   double a = System.currentTimeMillis();

   //System.out.println(Math.ceil((b-a)/1000/30));

   Après l'appel on calcule le nombre d'unités consommées, et on mit à jour le crédit
5. Ecrire la classe **Test.java** et instancier un objet Abonné.
6. Appeler setflexy(500) ,appeler(), afficherCrédit()

Abonné
-numéro : int -nom: String -crédit:float
+Abonné (String nom) +setNom() : void +getNom() : string +getCrédit() : float +setflexy() :void +appeler() : void +afficherCrédit() : void

Exercice 3 :

Une Voiture est caractérisée par sa marque, couleur, Matricule et Nombre de chevaux

1. Créer la classe voiture
2. Ajouter à la classe le constructeur qui permet d'initialiser la marque, la couleur, et le nombre de chevaux (le matricule est attribué automatiquement)
3. Encapsuler tous les attributs (les rendre private puis ajouter les accesseurs et modifieurs), l'attribut Matricule ne possède pas sa méthode modifieur en effet on ne peut pas modifier le matricule d'une voiture.
5. Ajouter une méthode **afficherTout()**
4. Dans la classe **Test.java** qui contient la méthode principale **main (),** créer un tableau qui contient cinq voitures puis remplir le tableau
9. Écrire le code qui permet d'augmenter le nombre des chevaux de 1 pour toutes les voitures du tableau
10. Afficher les informations des voitures après modification
11. Ecrire le code qui permet à l'utilisateur de saisir un matricule, puis afficher les informations de la voiture correspondante, si le matricule n'existe pas on affiche « voiture inexistante »

Exercice 4 :

Utiliser un numéro d'inscription personnalisé : D03/2011/213

```
Calendar c = Calendar.getInstance();
        int Anée_actuelle = c.get(Calendar.YEAR);
String a="D003/"+Integer.toString(Anée_actuelle)+"/"+ Integer.toString (numéro);
```

Exerice5:

```
class Compte
{
    //attributs
private int nombre;
private int numero;
private String nom;
private double solde;
    //définition des méthodes
public void Init(String UnNom)
    {
  numero = ++nombre;
  nom = unNom;
  solde = 0;
    }
public void Crediter(double Montant)
    {
  solde = solde + Montant;
    }
public void Debiter(double Montant)
    {
  solde = solde - Montant;
    }
public double GetSolde( )
    {
  return solde;
    }
}          //fin de la classe
```

Exercice 6:

On veut fabriquer une application qui permet de gérer une bibliothèque. Cette bibliothèque comporte plusieurs types de documents (classes) ; des livres, des magazines, mémoires de fin d'études

1. Décrire les différentes classes (attributs+méthodes avec leurs types)
2. Etablir un système d'héritage pur évité la répétition des caractéristiques communes

Exo3 (TP)

- **Méthodes** : **Constructeur Point(double X, double Y) getAbscisse**() et **getOrdonne**() qui
- **Méthodes :**
  . Un constructeur **Segment(Point A, Point B)** qui reçoit en entrée deux points
  . Une méthode **longueur()** qui retourne la longueur du segment

1. Créer une classe **Test.java** qui contient la méthode principale **main** (), puis instancier 2 objets point et un objet segment
2. Afficher la longueur du segment

Solution:

```
public class Test {

      public static void main(String[] args) {
Scanner sc1=new Scanner(System.in),sc2=new Scanner(System.in);

Point a,b;
System.out.println("entrez svp les coordonnées du premier point");
a=new Point(sc1.nextDouble(),sc1.nextDouble());
System.out.println("entrez svp les coordonnées du deuxième point");
b=new Point(sc1.nextDouble(),sc1.nextDouble());

Segment s=new Segment(a, b);
System.out.println(s.longueur());}
}
```

Exercice 2 (TP)

```
package serie2;
import java.util.Calendar;
public class employé {

	private String nom;
	private int ann_rec;
	private int nb_enf;
	private float salaire_base;

	public employé(String Nom,int Ann_rec,
int Nb_enf)
	{
	 nom=Nom;
	 ann_rec=Ann_rec;
	    if (Nb_enf>0)
		nb_enf=Nb_enf;
	}
	public String getNom()
	{
		return nom;
	}
	public void setNom(String Nom)
	{
		nom=Nom;
	}

	public int getNb()
	{
		return nb_enf;
	}
	public void setNb(int Nb)
	{if (Nb>0)
		nb_enf=Nb;
	}
	public float getSalaire_base()
	{
		return salaire_base;
	}
	public void setSalaire_base(float
Salaire_base)
	{
		salaire_base=Salaire_base;
	}
	public int getAnn_rec()
	{
		return ann_rec;
	}

}
********************
import java.util.Calendar;
public class Enseignant extends employé{
	private String module;
	private int nb_encadrés;


	public Enseignant(String Nom,int
Ann_rec, int Nb_enf,String Module,int
Nb_encadrés)
	{
		super(Nom,Ann_rec,Nb_enf);
		module=Module;
		nb_encadrés=Nb_encadrés;
		setSalaire_base(30000);
		}
	public int  getNb_encadrés()
	{
		return nb_encadrés;
	}
	public void setNb_encadrés(int
Nb_encadrés)
	{
		nb_encadrés=Nb_encadrés;
	}

	public String  getModule()
	{
		return module;
	}
	public void setModule(String Module)
	{
		module=Module;
	}

	public float  getSalaire()
	{ Calendar c = Calendar.getInstance();
	int Anée_actuelle =
c.get(Calendar.YEAR);
		float prime=(Anée_actuelle-
getAnn_rec())*800;
		float
impots=getSalaire_base()*0.18f;
		return
getSalaire_base()+getNb()*350+prime+nb_encadr
és*1000-impots;
	}
	public void affichertout()
	{
	System.out.println(getNom()+"
"+getModule()+"   "+getSalaire());
	}
}
**************
import java.util.Calendar;

public class Administrateur extends employé {

	private String service;

	public Administrateur(String Nom,int
Ann_rec, int Nb_enf,String Service)
	{
		super(Nom,Ann_rec,Nb_enf);
		service=Service;
		setSalaire_base(25000);
		}
	public String  getService()
	{
		return service;
	}
	public void setService(String Service)
	{
		service=Service;
	}

	public float  getSalaire()
	{ Calendar c = Calendar.getInstance();
	int Anée_actuelle =
c.get(Calendar.YEAR);
		float prime=(Anée_actuelle-
getAnn_rec())*800;
		float
impots=getSalaire_base()*0.18f;
		return
getSalaire_base()+getNb()*350+prime-impots;
	}
	public void affichertout()
	{
	System.out.println(getNom()+"
"+getService()+" "+getSalaire());
	}

}
```

Exercice : L'exercice suivant explique le principe du **polymorphisme**

```
public class employé {

	private String nom;
	private int age;
	private int nb_enf;
	private float heures;

	public employé(String Nom,int
Age, int Nb_enf,float Heures)
	{
	  nom=Nom;
	      age=Age;
	    nb_enf=Nb_enf;
	    heures=Heures;

	}
	public String getNom()
	{
	      return nom;
	}
	public void setNom(String
Nom)
	{
	      nom=Nom;
	}
	public int getAge()
	{
	      return age;
	}
	public void setAge(int Age)
	{if (Age>0)
	      age=Age;
	}
	public int getNb()
	{
	      return nb_enf;
	}
	public void setNb(int Nb)
	{if (Nb>0)
	      nb_enf=Nb;
	}
	public float getHeures()
	{
	      return heures;
	}
	public void setHeures(float
Heures)
	{
	      heures=Heures;
	}
```

```
	public void affichertout()
	{
		System.out.print(  "
"+getNom()+ " "+getAge()+ "
"+getNb());
	}

}

public class Enseignant extends
employé{
	private static int nombre;
	private int numéro;
	private String module;

	public Enseignant(int Age,
int Nb,String Nom,String
Module,float Heures)
	{

	super(Nom,Age,Nb,Heures);
		setModule(Module);
		numéro=++nombre;
	}
	public String  getModule()
	{
		return module;
	}
	public void setModule(String
Module)
	{
		module=Module;
	}

	public float  getSalaire()
	{
		return
800*getHeures()+getNb()*300;
	}

	public void affichertout()
	{	System.out.print(
numéro+" ");
		super.affichertout();
	System.out.println( "
"+getModule()+" "+getSalaire());
	}

}
public class Administrateur extends
employé {

	private static int nombre=0;
	private int numéro;
	private String service;

	public Administrateur(int
Age, int Nb,String Nom,String
Service,float Heures)
	{

	super(Nom,Age,Nb,Heures);
		setService(Service);
		numéro=++nombre;
	}
	public String  getService()
	{
		return service;
	}
	public void setService(String
Service)
	{
		service=Service;
	}

	public float  getSalaire()
	{
		return
500*getHeures()+getNb()*650;
	}
	public void affichertout()
	{	System.out.print(
numéro+" ");
		super.affichertout();
	System.out.println( "
"+getService()+" "+getSalaire());
	}

}


public class Test {
	public static void
main(String[] args) {
employé[] tab = new employé[4];
tab[0]=new
Enseignant(30,2,"AA","analyse",15.5
f);
tab[1]=new
Administrateur(50,4,"BB","comptabil
ité",35);
tab[2]=new
Enseignant(45,3,"CC","informatique"
,80);
tab[3]=new
Administrateur(35,0,"DD","bibliothè
que",40.5f);

for (int i=0;i<tab.length;i++)
	tab[i].affichertout();

	}
}
```

Les packages en java

- En Java, un package est une collection (regroupement) de classes. Il correspond aux « bibliothèques » des autres langages
- Pour créer un package, il suffit de commencer le fichier source contenant les classes à regrouper par l'instruction *package* suivi du nom que l'on désire donner au package (le nom du projet)
- Charger un package nous permet d'utiliser les classes qu'il contient.
- Parmi les packages les plus utilisés, on peut citer les suivants :

Package	Description
java.awt	Classes graphiques et de gestion d'interfaces
java.io	Gestion des entrées/sorties
java.lang	Classes de base (importé par défaut)
java.util	Classes utilitaires
javax.swing	Autres classes graphiques

- Une seule classe du package est importée :

import java.util.Scanner ;

- toutes les classes du package sont importées (même les classes non utilisées) :

import java.util.* ;

- Les classes déclarées public sont visibles depuis l'extérieur du package qui les contient.
- Les classes n'ayant pas été déclarées public ne seront accessibles que depuis l'intérieur du package : on dit que leur portée est default.
- Une classe dans le même package est importé automatiquement (sans import)

Remarque :

1. ***import static java.lang.Math.*;*** toute la classe Math, alors on peut appeler sin(x) directement
2. pour importer un package dans un projet :
   Proprietés/java build path/librairies/add class folder/cocher bin